1972

A Methodology for the Design and Optimization of Information Processing Systems

J. F. Nunamaker

Report Number:
72-051

Nunamaker, J. F., "A Methodology for the Design and Optimization of Information Processing Systems" (1972). *Department of Computer Science Technical Reports*. Paper 431.
<https://docs.lib.purdue.edu/cstech/431>

A METHODOLOGY FOR THE DESIGN AND OPTIMIZATION
OF INFORMATION PROCESSING SYSTEMS

J. F. NUNAMAKER, JR.

CSD TR 51

A METHODOLOGY FOR THE DESIGN AND OPTIMIZATION
OF INFORMATION PROCESSING SYSTEMS*

J. F. Nunmaker, Jr. **

Department of Computer Sciences
412 Mathematical Sciences Building
Purdue University
Lafayette, Indiana 47907
Telephone: 317/493/3832
317/749/3566

*The work described in this paper was sponsored in part by the IHDOS (Information Systems Design and Optimization System) Project, A University of Michigan Research Program directed by Professor Daniel Teichrow and by a Krannert Research Grant at Purdue University.

**Assistant Professor of Computer Sciences and Industrial Administration. This is a working paper. Comments are invited and should be directed to the author at the above address. Please do not reproduce or quote without the author's permission.

A Methodology for the Design and Optimization
of Information Processing Systems

J. F. NUNAMAKER, JR.

ABSTRACT

The Information Processing Systems design problem is examined with the objective of developing formal techniques which will be of use to analysts and eventually will result in automating the systems design functions.

An overview of the Purdue version of SODA (Systems Optimization and Design Algorithm) is presented. The implementation of an earlier version of SODA written for the UNIVAC 1108 at Case Western Reserve University is discussed.

SODA is formulated as a multilevel decision structure utilizing optimization techniques (graph theory and mathematical programming) and heuristic procedures. The objective of SODA is to generate a complete systems design starting from a statement of the processing requirements. The output of SODA consists of specifications for the selection of hardware (CPU, core size, auxiliary memory and input/output configuration), the specification of program modules, files, and system scheduler.

SODA consists of a problem statement technique (PSL), problem statement analyzer (PSA), a procedure for the selection of alternative hardware configurations and the generation of alternative designs of program structure and file structures (ALT), and a procedure for the optimization and performance evaluation of each systems design (OPT).

SODA is limited to the design of uni-programmed batch systems, sequential auxiliary storage organization, the specification of linear data structures, and the selection of a single CPU. The model is deterministic. The problem statement technique is intended to handle "report oriented" data processing systems.

INTRODUCTION

The design of an Information Processing System (IPS) can be divided into two major problems:

1. What are the requirements of an information system, e.g., what outputs should be produced?
2. What is the best way to produce the required outputs, on time, given the requirements developed in 1?

This paper is concerned with the second problem. However, the methodology developed to solve the second problem provides cost information that can be used by the problem definer to assist him with the determination of the requirements. One approach to determining the requirements of an information system involves comparing the information value of a report with the cost

of producing the report.*

Considerable time and money are expended in system design and programming when a firm acquires, leases or in any way uses a computer. With each new system the task of getting the system operational seems to take longer and becomes more costly than the time before. A methodology for analyzing and designing Information Processing Systems is needed if we are to keep from getting further behind.

The general purpose of this paper is to continue the formalization of the process for designing Information Processing Systems and to improve it by the increased application of operations research techniques and by more use of the computer itself.

SODA (Systems Optimization and Design Algorithm) is presented as a methodology for automating the system design functions. The objective of SODA is to generate a complete systems design starting from a statement of the processing requirements.

INFORMATION PROCESSING SYSTEMS

An Information Processing System is here defined as a set of personnel, hardware, software packages, computer programs and procedures that have been assembled and structured so that the whole set accomplishes some given data processing requirements in accordance with some given performance criterion.

An important aspect of this definition is that it includes an explicit statement of the "performance criterion" by which performance of the system is measured. A consequence of including performance measures is that the emphasis is focused on the over all performance of the system rather than on any one part. The study of large scale IPS is in essence a study of the performance of the total system: hardware, software and other procedures.

One characteristic of an IPS is that data files are stored on auxiliary memories and it takes a number of interrelated computer programs to meet the specific requirements of the problem definer. The large number of interrelated programs distinguishes the problem that is described here from that aspect of Computer Science which is concerned with individual programs. These systems almost always depend on a large amount of data, now frequently called a data base. A duality exists between the programs and the data, and the structure of each is quite important.^{1,2}

The selection of expensive hardware for a given set of requirements is

* The value of a report is often arrived at by a process of "guesstimation" and it is difficult to accurately measure the value.

frequently involved and often the expenses increase significantly since the requirements of the system are continuously changing.

What is needed is a flexible systems design process that can accommodate changing requirements.

THE IPS DESIGN PROCESS

The IPS design process has a number of similarities to any physical design process, such as a production plant or a bridge. In each case there must be an initial recognition of a need. Next, preliminary studies are conducted in which major alternatives are considered, the technical feasibility determined and costs of alternatives estimated. If a decision to proceed is made, the requirements must be stated in sufficient detail for designing the system. The design phase consists of preparing a set of specifications (blueprints) which are detailed enough for the construction phase.

The major functional activities and decision points in the design process of IPS are shown in Figure 1. The design process is initiated through the statement of requirements from "Problem Definers." After the requirements have been documented, the systems analysts consider the equipment available and any constraints (such as the existing system) on the design activity. The design phase consists of producing the specifications for the four major parts of the system:

- Hardware and software packages that will be used
- Programs to be written
- System Scheduler, schedule for sequencing the running of the programs
- Data Organization, specifications of file structure and how the files will be stored in hardware memories

Figure 1. The Systems Design Process:

Beneath the surface similarities between the design process of physical structures and IPS there are some differences in emphasis. Typically, more attention is given to the planning and generating alternative designs in developing a production facility than in development of an IPS. This is probably true because there are more external constraints associated with the design of a new facility, e.g., architects, contractors, equipment suppliers, and governmental zoning commissions. In the design and implementation of the Information Processing System, the requirements for the formalization of the design functions are not so apparent, hence there is a tendency to do some of the design work concurrently with construction of the system. This practice often leads to problems.

External constraints, similar to those involved in the physical design process, can be created to formalize the requirements of the Information Processing Department (IPD) through the use of information budgets.³ The information budget will directly involve management in the operation of the IPD and force more attention to be given to the IPS design.

CURRENT PRACTICE IN SYSTEMS DESIGN

While some formal techniques have been proposed and the computer is sometimes used for calculating estimated processing time, most of the systems design is done in an ad hoc basis. The need for formal analysis techniques, of course, has long been recognized.^{1,2,4,5}

Information Processing Department managers generally recognize a distinction between systems analysts and programmers. The systems analyst is usually responsible for systems design. In most cases he has had no formal education or training for systems design and has obtained his knowledge by experience. He uses little in the way of tools other than graphical communication devices such as flow charts and decision tables.

Complicating the problems of inexperience and lack of training of the systems analyst is the fact that IPS problems must be subdivided to reduce the task for a systems analyst to a reasonable size and this introduces the problem of coordination of the many systems analysts on the project. In addition it involves the coordination of problem definers, systems analysts and programmers.

The systems design is carried out by one or more analysts who obtain the statement of requirements from those who specify what processing is to be done and what output is needed. The analysts specify programs and file design in sufficient detail so that programmers can write the programs and the files can be constructed.

There is considerable difference of opinion on how much detail of the systems design should be documented and how much other communication between systems analysts and programmers should be allowed. Despite all these problems, information processing systems are being designed.

There are, however, some major undesirable features of the present procedures for systems design. These are:

1. The performance criteria and the requirements of the IPS are not stated explicitly.
2. Programs become the only up-to-date documentation.
3. Accomodation of changes to the IPS is expensive.
4. The design process takes too long.
5. Construction of the system frequently starts before the system is completely designed in an effort to save time.
6. Few alternatives are examined in the design phase.
7. The systems do not work correctly.
8. They are costly to design.
9. Procedures become inefficient as changes in the IPS occur.

Optimization has not been completely overlooked in the design and implementation of Information Processing Systems, but any such effort has been applied to the evaluation and selection of equipment usually for a specified application. This should not be surprising since the commitment for computing system hardware represents a sizable outlay. Since a contract is to be considered, it represents a decision point in which management becomes directly involved. Because of this, care is given to the consideration of alternatives to assure the "best decision" is made--perhaps too much care, in relationship to the return that can be expected from review focused only on this decision point.

The difficulties both of time lag and problem expression summarized above are familiar to anyone who has been involved in management of a large scale Information Processing System department. In addition to these existing difficulties, the design problem will become even more difficult in the future. The hardware is already able to accommodate more powerful software systems than are available. Moreover, attempts to develop sophisticated software systems have been very expensive.

The first three points listed above are related to the deficiencies in problem definition and documentation. Points four and five are concerned with the time required to carry out the systems design and the sixth point relates to the lack of optimal use of resources. Points seven, eight and nine

are concerned with the realization that the systems often do not work properly. They are very costly and become inefficient over time.

THE IPS DESIGN PROBLEM

It is becoming more and more apparent that the drawbacks listed in the previous section are likely to become more of a problem in the future. The design of IPS to handle more complex requirements cannot be achieved without unrealistic expenditures of effort. The number of analysts, designers and programmers required to handle these more complex requirements are not likely to be available.

There are basically four ways to improve the situation:

1. Education to increase the number of personnel and improve their quality.
2. Improvement of manual system design tools, techniques, and procedures.
3. Use of generalized rather than tailor-made software.
4. Automation of the system design process.

Most attention has been devoted to the first three approaches. The SODA methodology concentrates on the fourth approach and rests on the premise that specifications for the IPS can be generated directly from a statement of user requirements for a limited class of processing requirements. The specifications must be detailed enough to verify feasibility and to evaluate the performance of the proposed system but not more detailed than enough to specify construction because producing "too detailed specifications" are costly and they may have to be changed in any case. Also, "too detailed specifications" are embedded in processing procedures which tends to bind the design unnecessarily, at too early a stage and with negative payoff.

Since the purpose of the IPS is to produce outputs, it must respond to changing inputs. One of the design problems is to decide what changes should be accommodated with what degree of ease. Certain components are easier to change than others, e.g., it is easier to change a program than to change the operation performed in the hardware. It is easier to change the data in a file than it is to change the structure of the file.

The systems design decision hierarchy consists of certain decisions which constrain later activities. For example:

1. Selecting a hardware configuration constrains everything that follows to what can be accomplished with the selected hardware.
 2. Selecting the size of the main memory, number and type of input and output units and auxiliary memory constrains still further what can be accomplished.
-

3. Selecting an operating system determines what processing organization is possible.
4. Selecting compilers and utility programs constrains what the programmer can do for himself. The programmer, when he writes a particular program module, is constrained by the data organization, the input and output formats and the software and hardware on which the program is run.

The purpose of our discussion of design is to identify the design decisions that are made at any point, to enumerate the decisions that are possible, and to develop methods for:

1. Determining when (in the design process) the decisions should be made.
2. What is the optimal or suboptimal decision in a particular case.

Currently each organization tends to develop its own procedures with very little evaluation of methods developed elsewhere. IPS design can benefit by the synthesis of the available knowledge and practical know-how, and by the development of more powerful analytical methods to replace the current ad hoc methods.

In the course of the evaluation of computers and their use as information processing devices a variety of tools have been developed to aid the analyst in making the necessary design decisions and in facilitating the construction of these systems. Steiger⁶ describes some of the steps in systems design and it is clear that as these aids have become more sophisticated the computer is being used more and more extensively in the design process. For example, there are commercial computer systems simulation packages^{7,8,9} available for use in the evaluation of computer system performance.

However, most of the use of the computer has been in the construction phase, i.e., in the generation of computer code from source language statements. These source language statements could only be prepared once the system had been designed, i.e., once the analyst had decided what hardware would be used, how data would be organized and stored in the hardware, how the processing would be combined into programs, and how these programs would be organized, i.e., in what order the programs would be run to accomplish the total processing requirements. It is exactly these decisions which have a major influence on the performance of the system, i.e., how much computer hardware and how much time are needed to satisfy a given set of processing requirements.

SODA

It is with respect to systems design decisions that much work needs to be done and SODA is presented as a methodology for the design and optimization of IPS. The existing systems simulators^{7,8,9} assume as given a systems design,

i.e., a description of each program, schedule for a set of runs and structure of the data files. SODA is intended to specify a systems design from a statement of the requirements and to generate the set of programs and data files.

SODA consists of a number of sub-models that are solved using mathematical programming, graph theory and heuristic procedures. Since the overall design problem is very large, it is convenient to view the algorithm as a multilevel decision model with the decision variable of one level becoming a constraint at the next level and so on. The partitioning of the problem into a multilevel structure implies that a different set of decision variables are required for each level of the algorithm.

The decision making structure of SODA is described by (1) specification of the inputs and outputs, (2) specification of decision variables and determination of feasible alternatives, (3) selection of an objective function, (4) expression of objectives as a function of decision variables, (5) explicit statement of constraints which limit the value of the decision variable and (6) solution, i.e., determination of the values of the decision variables.

SODA is a set of computer programs which begins with the initial statement of requirements (i.e., what the system is to do) and proceeds through the design and specification of the system. SODA is not concerned with the determination of which requirements are to be stated. The assumption is made that the problem definer (PD) can accurately identify his requirements. The major components of SODA are:

Problem Statement Language (PSL)

SODA/PSL is a technique for stating the requirements of the IPS independent of processing procedures. It also provides the capability for easily handling changes in requirements.

Problem Statement Analyzer (PSA)

SODA/PSA is a program for analyzing the statement of the problem and organizing the information required in SODA/ALT and SODA/OPT. This program also provides feedback information to the problem definer to assist him in achieving a better problem statement.

Generation of Alternative Designs (ALT)

SODA/ALT is a procedure for the selection of a CPU and core size and the specification of alternative designs of program structure and file structure.

Optimization and Performance Evaluation (OPT)

SODA/OPT is a procedure for the selection of auxiliary memory devices and

the optimization and performance evaluation of alternative designs.

Refer to Figure 2 for an overview of SODA.

Figure 2. SODA: Systems Optimization and Design Algorithm

The output of SODA is (1) a list specifying which of the available computing resources will be used (2) specifications of the programs generated (3) specifications of the file structure and the devices on which they will be stored and (4) a schedule of the sequence in which the programs must be run to accomplish all the requirements.

SODA selects a set of hardware, generates a set of programs and files that satisfy timing requirements, core memory and storage constraints such that the hardware cost of the system is minimized.

SODA is limited to the design of uni-programmed batch systems, sequential auxiliary storage organization, the specification of linear data structures, and the selection of a single CPU. The model is deterministic. The problem statement technique is intended to handle "report oriented" data processing systems. Refer to Figure 3 for the interaction of the levels of SODA.

Figure 3. Interaction of the Levels of SODA

The overall structure of SODA/ALT and SODA/OPT is given in Figure 4 which describes the decision variables, objective function, alternatives, constraints and solution techniques for each level in a summary form.

<u>Level</u>	<u>Decision Variables</u>	<u>Objective</u>	<u>Alternatives</u>	<u>Constraints</u>	<u>Optimization Techniques</u>
SODA/ALT					
1	CPU	Select the minimum rental cost CPU class	list of available CPU classes	Time available for processing	search for ordered CPU classes
2	Core Size	Select the minimum rental cost core size	list of available core sizes	-Times available for processing -CPU class	search of ordered core sizes
3	Program Module	Select the set of modules with the minimum transport volume	feasible grouping arranged in tree structure	-Core Size available for modules	Network analysis and branch and bound search over feasible alternatives
4	Data Structures	Select Files such that the maximum number of I/O for any module is a minimum	feasible grouping arranged in tree structure	-Program Modules	Network analysis and branch and bound search over feasible alternatives
SODA/OPT					
1	Storage Structure	Minimize the number of inter block gaps for tape or the number of accesses for a disc	1 char $\leq$ block size $\leq$ upper limit 1 $\leq$ accesses $\leq$ upper limit	-Program Modules -Data structure	Non-linear programming model
2	Number and type of auxiliary memories	Minimize the variable reading and writing time	list of available devices	-CPU and Core Size -Program Modules -Storage structures -Data structure	Integer Programming Model

Figure 4. Decision Levels of SODA

PROBLEM STATEMENT LANGUAGE

It is assumed that someone called a Problem Definer (PD) is familiar with the operation of the organization and has the necessary training to describe the processing requirements of the organization. The Problem Definer states his data processing requirements in a problem statement (PS) according to SODA/PSL and the requirements are input to SODA/PSA in the form of a subset of a PS called a Problem Statement Unit (PSU). A PSU consists of three major categories: the data description, processing requirements and operational requirements. The data description is defined by Elementary Data Sets (eds) and Data Sets (ds). The processing requirements consist essentially of a set of formulas called Processes (pr). The operational requirements consist of information on volumes, frequency of output and timing of input and output.

An eds consists of a Data Name (dn), Data Value (dv), Descriptor Name (sr) and Descriptor Value (sv).

The sales of model X in the north region is an example of an eds.

<u>Data Name</u>		<u>Descriptor Value</u>	<u>Descriptor Name</u>		<u>Data Value</u>
SALES_MODEL_X	(in the)	NORTH	REGION	(is)	500

A Data Set is the set of all eds with the same dn. The sales of model X in all regions of the country is an example of a Data Set.

<u>Data Name</u>		<u>Descriptor Value</u>	<u>Descriptor Name</u>		<u>Data Value</u>
SALES_MODEL_X	(in the)	NORTH	REGION	(is)	500
SALES_MODEL_X	(in the)	SOUTH	REGION	(is)	600
SALES_MODEL_X	(in the)	EAST	REGION	(is)	300
SALES_MODEL_X	(in the)	WEST	REGION	(is)	400

There are four types of Data Sets and SODA makes use of the Data Set type in the file structure algorithm. An Input Data Set is any input data to the IPS. A Storage Data Set is that data which is stored in the IPS. A Terminal Data Set consists of output reports or forms and is not retained in permanent storage in the IPS. A Computed Data Set is the output of a Process that is neither a Storage nor a Terminal Data Set.

A Data Set may have up to three Descriptor Pairs to uniquely identify a Data Value. The set of Descriptor Pairs is referred to as an Identifier and a Descriptor Pair consists of a Descriptor Name and Descriptor Value.

Let id_k be the k^{th} Identifier where $dn[id]$ is a shorthand notation for writing $\langle dn, sv \rangle, \langle sv, dv \rangle$.

Example: The GROSS_PAY of 5 employees each identified by the Identifier consisting of a single Descriptor EMPLOYEE_NUMBER is written as

<GROSS_PAY [id₁]> and is equivalent to:

<GROSS_PAY, sn₁>, <sv₁, sv₂ . . . sv₅, dv₁, dv₂ . . . dv₅>

where:

sn₁ = EMPLOYEE_NUMBER

sv₁ = 17 dv₁ = \$263

· ·

· ·

sv₅ = 21 dv₅ = \$300

A Process is the smallest unit of processing requirement that may be grouped, but never subdivided by SODA. A Process produces a Data Set and is a well defined assignment type statement. There are four types of Processes defined in SODA/PSL. They are: (1) COMPUTE, (2) SUM, (3) IF and (4) GROUP.

The Compute Process is a variation of the familiar assignment statement found in FORTRAN, ALGOL and COBOL.

An example of the COMPUTE Process is:

NET_PAY [id₁] = COMPUTE GROSS_PAY [id₁] - DEDUCTIONS [id₁]

This expression consists of three Data Sets, NET_PAY [id₁], GROSS_PAY [id₁], DEDUCTIONS [id₁] and all are identified by id₁ (EMPLOYEE_NUMBER). It is assumed that whenever the Process is to be executed it is executed for all valid Descriptor pairs. The number of Descriptor pairs and the number of each type of arithmetic operation (e.g., multiplication, addition) contained in a Process is used by SODA/ALT for the purpose of estimating running times. The number of each type of arithmetic operation is obtained from the list of Processes.

The SUM Process is an expression that makes it convenient to sum over two or three Descriptor Names and the associated Descriptor Values. For example, one could sum labor cost by employee numbers and department numbers. The IF Process is a conditional expression. The GROUP Process groups Data Sets that are required to represent output reports. The four types of Processes enables SODA/PSA to construct precedence graphs of the Processes and Data Sets that are necessary for the Program and File Structure Algorithms.

The time and volume characteristics of the IPS are also described in the problem statement. Time requirements are specified by stating absolute time deadlines, i.e., paychecks must be produced by 4:00 PM on Friday. The statement of the time requirements for the output reports of the IPS is expressed

by a Need Vector. The Data Set volumes are computed from the volumes stated for each eds. All time and volume information is expressed in units specified by the problem definer.

The structure of a Problem Statement in SODA/PSL consists of:

Problem Statement Name

List of Identifiers

List of Descriptors

Descriptor Name

The Number of Descriptor Values for each Descriptor Name

List of Data Sets

Data Name

Volume of Data Set

Type of Data Set

List of PSU

Contents of each PSU

PSU Number

PSU Name

Need Vector

List of Processes

END of PSU

END OF PROBLEM STATEMENT

SODA/PSL is a non-procedural Problem Statement Language in the sense that the PD writes a PSU without imposing any procedural ordering on the Processes. The precedence relationships of the Processes are inferred by SODA/PSA.

The PS must contain sufficient detail so that systems analysts and programmers could use it (if necessary) to design and implement the Information Processing System with no additional information.

PROBLEM STATEMENT ANALYZER

The problem statement analyzer (SODA/PSA) accepts the requirements stated in SODA/PSL, analyzes them and provides the problem definer diagnostics for debugging his problem statements and reports. SODA/PSA also produces a number of networks which record the interrelationships of Processes and data and passes the networks on to SODA/ALT.

Each type of input and output is specified in terms of the data involved, the transformation needed to produce output from input and stored data. Time and volume requirements are also stated. SODA/PSA analyzes the statement of the problem to determine whether the required output can be produced from the available inputs. The PS stored in machine readable form is processed by SODA/PSA which:

1. checks for consistency in the PS and checks syntax in accordance with SODA/PSI; i.e., verifies that the PS satisfies SODA/PSL rules and is consistent, unambiguous, and complete.
2. prepares summary analyses and error comments to aid the problem definer in correcting, modifying and extending his PS, and
3. prepares data to pass the PS on to SODA/ALT.
4. prepares a number of matrices that express the interrelationship of Processes and Data Sets.

There are a number of papers that discuss the use of graphs and their associated matrices for the analysis of program and data structure^{1,2,10,11,12,13}.

SODA/PSA follows the papers of Borje Langefors^{1,2} and Raymond B. Briggs.¹⁰ Langefors discusses the use of matrix algebra and graph theory to represent the processing units and data units in an IPS. Langefors' work differs from others using graph theory for this purpose in that it includes a performance criteria to be optimized.

Briggs added to the matrix definitions of Langefors and provided the necessary structure to develop a Program and File Structure Algorithm.

The problem statement is defined in SODA/PSA as the set of Processes required, the set of Data Sets needed by each Process and the precedence relationships of the Processes (pr) and Data Sets (ds).

SODA/PSA generates the P, P* and E matrices for each PSU and for the entire IPS.

P - Precedence Matrix: Data Sets

$P_{ij} = 1$ if ds_i is a direct precedent of ds_j

$P_{ij} = 0$ otherwise

P* - Precedence Matrix: Processes

$P^*_{ij} = 1$ if pr_i is a direct precedent of pr_j

$P^*_{ij} = 0$ otherwise

The precedence matrices are checked for consistency using Marimont's procedure.¹⁴

E - Incidence Matrix: Processes and Data Sets

$e_{ij} = 1$ if ds_j is an input to pr_i

$e_{ij} = -1$ if ds_j is an output of pr_i

$e_{ij} = 0$ if there is no incidence between ds_j and pr_i

Let v_j be the volume of ds_j , t_i be the number of inputs and outputs for each Process and m_j be the multiplicity of Data Set transport for ds_j . Let n_j represent the number of times ds_j is an input or output of a set of Processes.

$$t_i = \sum_{j=1}^k |e_{ij}| ; i=1,2,\dots,n.$$

$$m_j = \sum_{i=1}^n |e_{ij}| ; j=1,2,\dots,k.$$

The transport volume for ds_j is

$$tv_j = m_j \cdot v_j$$

The transport volume for the set of Data Sets is

$$TV = \sum_{j=1}^k m_j \cdot v_j$$

Transport volume is used as a criterion to evaluate alternative program and data designs and is discussed in the next section.

Let ds_j be represented by a and pr_i be represented by a . An example of an incidence graph and the associated incidence matrix is given in Figure 5.

The associated incidence matrix is:

		Data Sets					t_i
		a	b	c	d	e	
Processes	A	-1	1	1	0	0	3
	B	0	-1	1	0	0	2
	C	0	0	-1	1	1	3
m_j		1	2	3	1	1	
v_j		20	20	20	20	20	
tv_j		20	40	60	20	20	

The transport volume for the Data Sets (TV) in this example is 160 units.

Figure 5. Incidence Graph and Matrix

The R, R* and M matrices are generated for the entire set of Processes*.

R - Reachability Matrix: Processes

The R matrix is used to check precedence violations in the grouping procedure of SODA/ALT.

$$R = (P^*) \vee (P^*)^2 \vee \dots \vee (P^*)^{q-1}$$

where q is the index of the nilpotent matrix P*

$r_{ij} = 1$ if pr_i has any precedence relationship with pr_j

$r_{ij} = 0$ otherwise.

R* - Partial Reachability Matrix: Processes

The R* matrix is used to calculate the M matrix.

$$R^* = (P^*)^2 \vee (P^*)^3 \vee \dots \vee (P^*)^{q-1}$$

$r^*_{ij} = 1$ if pr_i has a higher (2 or more) order precedence with pr_j .

$r^*_{ij} = 0$ otherwise.

It was shown by Briggs¹⁰ that by using a theorem proved by Warshall¹¹

that R and R* can be constructed without first computing successive powers of P*.

M - Feasible Process Grouping Matrix: Processes

If $m_{ij} = -1$ there exists higher (2 or more) order relationships between pr_i and pr_j and pr_i cannot be combined with pr_j . If $m_{ij} = 0$ there is no precedence ordering and pr_i can be combined with pr_j . This indicates a feasible but not necessarily profitable grouping. If $m_{ij} = 1$ there is a direct precedent relationship and pr_i can and should be combined with pr_j since this indicates a feasible and profitable grouping. If $m_{ij} = 2$ there is an immediate reduction in logical input-output requirements when pr_i and pr_j are grouped.

$m_{ij} = -1$ if $r^*_{ij} = 1$ or $r^*_{ji} = 1$

$m_{ij} = 0$ if $r^*_{ij} = 0$ and $r^*_{ji} = 0$ and $p^*_{ij} = 0$ and $p^*_{ji} = 0$; except when $p^*_{il} = 1$ and $p^*_{jl} = 1$ or $p^*_{li} = 1$ and $p^*_{lj} = 1$.

$m_{ij} = 1$ if $r^*_{ij} = 0$ and $r^*_{ji} = 0$ and $p^*_{ij} = 1$ or $p^*_{ji} = 1$

$m_{ij} = 2$ if $r^*_{ij} = 0$ and $r^*_{ji} = 0$ and $p^*_{il} = 1$ and $p^*_{jl} = 1$ or $p^*_{li} = 1$ and $p^*_{lj} = 1$

pr_l has a first order precedence or succedent relationship with pr_i and pr_j .

A list of all feasible pairs for grouping of Processes is constructed from the M Matrix and passed to SODA/ALT.

*A procedure is discussed briefly in the next section for partitioning the entire set of Processes into smaller groups when the number of Processes is very large.

GENERATION OF ALTERNATIVES

The information system design phase begins after the requirements have been stated, verified and analyzed in SODA/PSA. SODA/ALT accepts as input, the output of SODA/PSA and a statement of the available computing resources, hardware and utility programs. The hardware alternatives are ordered in a tree structure as shown in Figure 6.

Figure 6. Hardware Alternatives

A feasible CPU and core size are specified using a heuristic timing procedure. An ordered search of different CPU's is made in an attempt to find the minimum CPU assuming an infinite core memory with no auxiliary memory and all Processes and Data Sets in real core. The premise is that if a CPU cannot perform adequately under these "ideal" conditions it cannot possibly be adequate with limited core constraints.

The processing time for each time period (i.e., a week) is computed. If this is less than the actual time available to do this, then a lower bound of

CPU capability is found. If not, the next CPU is tried. If it appears that some shifting of load from one or more time periods (i.e., week 1) to other time periods (i.e., week 3) could solve the problem, then the problem definer is advised about it and given a chance to "level" the requirements.

Having found a CPU that will perform adequately under infinite core assumptions, an ordered search (starting with the smallest core size) is made of available core sizes for this CPU.

Using as constraints CPU and core size, a graph theoretic model generates alternative Program Module and File Designs. A Program Module (pm) is a set of Processes grouped together by SODA/ALT. A File (f) is one or more Data Sets that are grouped together. From the Matrix of feasible groupings (M) a list of feasible (profitable) groupings of Processes is obtained. The list of feasible groupings is partitioned into 3 cases to reduce the number of alternative program designs that must be evaluated. The cases separate output reports into classifications of due dates for reports. The cases are then divided into subcases. (A subcase is a group of Processes that has no Process precedence link to other subcases.) For each subcase, feasible Program Modules of size 3, 4, . . . N are generated where N is the number of Processes in the partitioned list (or sub case).

It is known that by grouping Processes into a composite process called a Program Module, the multiple input and output of Data Sets can be reduced. Such grouping of processes, however, requires additional main memory for the Program Modules. In generating an efficient design, it is necessary to decrease the transport volume (total number of characters read in and written out of main memory) in order to reduce the processing time. If Data Set volumes remain constant, in order to decrease the transport volume, the multiplicity (the number of times a data set is input and output) of Data Set transport must be decreased. After the Program Modules are specified the data sets are consolidated into Files for the purpose of reducing the number of input/output Files required and for better utilization of storage in auxiliary memory. Process grouping is shown to correspond to a grouping of rows of the incidence matrix, and data set consolidation is shown to correspond to a grouping of columns.

Program Module and File Design is concerned with the reduction of processing time and can be summarized by the two methods by which the processing time can be reduced. SODA/ALT determines:

1. which operations (Processes) will be grouped into Program Modules
 - a. Group Processes which will eliminate the writing out and reading in of a Data Set. For instance Figure 7a.
 - b. Group Processes which require the same Data Sets. For instance Figure 7b.

The transport volume of ds_a is eliminated when pr_A and pr_B are grouped.

The transport volume is reduced when pr_A and pr_B are grouped since ds_c is read only once.

and it may be profitable to also group ds_a and ds_b .

Figure 7. Methods for reducing processing time

The objective is to reduce total transport volume and thus total processing time.

2. which Data Sets will be grouped into Files (f)
 - a. organize the data structure of the Files so that data which are needed together are close together in order to reduce searching time.
 - b. organize the data structure of the Files such that fewer logical input/output devices are needed.

The following matrices are used in SODA/AIT to generate alternative Program Module and File Designs

S - Program Module Selection Matrix: Program Modules and Processes.

The S matrix represents the alternative grouping of Processes.

$s_{ij} = 1$ if pr_j is a member of pm_i

$s_{ij} = 0$ otherwise.

E' - Incidence Matrix: Program Modules and Data Sets

The S matrix is multiplied by the E matrix to produce the new incidence matrix E' , where $E' = SAE$

The Boolean matrix operators " $\wedge$ " and " $\vee$ " for Process grouping follows the rules of Boolean Algebra with the following exceptions for the Boolean addition operator.

$0 \vee -1 = -1$

$1 \vee -1 = 0$ if the output ds_j is used only by the pm_i with which it is grouped and ds_j is not a Terminal DS or Storage DS.

$1 \vee -1 = 1$ if the output ds_j is required in a Process that is a member of another Program Module or is a Terminal DS or Storage DS.

D - Feasible Data Set Grouping Matrix: Data Sets

$d_{ij} = 1$ if $e'_{ij_\alpha} = 1$ and $e'_{ij_\beta} = 1$ or $e'_{ij_\alpha} = -1$ and $e'_{ij_\beta} = -1$ and ds_{j_α} and ds_{j_β} are the same data types and have a common descriptor or if ds_i and ds_j are input Data Sets and have a common descriptor.

$d_{ij} = 0$ otherwise.

ds_{j_α} and ds_{j_β} are two data sets required by pm_i . The same data type for ds_{j_α} and ds_{j_β} refers to the classification of Input ds, Computed ds, Storage ds and Terminal ds. The test for a common Descriptor Name is intended to link Data Sets together that have a relationship other than common input or common output of a Program Module.

G - File Selection Matrix: Data Sets and Files

The G matrix represents the alternative grouping of Data Sets.

$g_{ij} = 1$ if ds_i is a member of f_j

$g_{ij} = 0$ otherwise

E' - Incidence Matrix: Program Modules and Files

The E' matrix is multiplied (as described earlier) by the G matrix to produce the incidence matrix E'' of Program Modules and Files, where

$$E'' = E'AG$$

The selection procedure for program design is organized as a tree structure with all feasible alternatives ordered in terms of core memory requirements and transport volume. The procedure for File design is organized by descriptors (keys) and the number of input/output Files required for each Program Module.

If software modules such as sort modules are required to process Files they are inserted in the IPS Design.

The next step is to look for design improvements and to select a specific number and type of auxiliary memory units.

OPTIMIZATION AND PERFORMANCE EVALUATION

The optimization and performance evaluation phase generates a storage structure and scheduler, selects auxiliary memory devices, and searches for ways to improve the IPS design. SODA/OPT may return control to SODA/ALT to select another CPU, core size or to select another set of Program Modules and Files.

SODA/OPT selects the minimum cost hardware configuration that is capable of processing the stated requirements in the time available. This phase consists of a number of mathematical programming models and timing routines that are used to (1) optimize the blocking factors for all Files, (2) evaluate alternative designs; i.e., specify the number and type of auxiliary memory devices, (3) assign Files to memory devices and (4) generate an operating schedule for running program Modules. These sub-models follow the work of Schneidewind¹⁶ and Thiess¹⁷. Refer to McCuskey¹⁸ for another approach to the design of data organization.

In SODA/OPT the performance criterion is optimized within the constraint set by the capability of the hardware and by the processing requirements. SODA/OPT produces a report describing the system and stating its predicted performance. On the basis of this, the Problem Definer may decide to change his PS, or accept the design; SODA/OPT then provides detailed specifications for the construction of the system.

The output of SODA/OPT is:

1. A list specifying which of the available computing resources will be used.
2. A list of the Program Modules specifying the input, output and computations to be performed in each.
3. A list of Files to be assigned specifying their format and manner in which they will be stored. Assignment of Files to memory devices.
4. A statement of the sequence and manner in which the Program Modules must be run to accomplish all the requirements.

IMPLEMENTATION

SODA has been written for the Univac 1108 in FORTRAN at Case Western Reserve University¹⁹; the program has also been implemented on the IBM 360/57 at the University of Michigan by Professor Daniel Teichroew. The program is currently being modified and rewritten for the CDC 6500 at Purdue University²⁰.

The Case version of SODA has the following drawback: If there are two feasible solutions

- (1) Small CPU, large core
- (2) Larger CPU, smaller core

such that (2) is cheaper, then the algorithm would not find (2). The assumption is made (the Case version) on the ordering of the hardware tree that this situation (2) would not occur. This was done in order to simplify the search procedure and to reduce the size of the large combinatorial problem involved. A branch and bound procedure is being implemented in the Purdue version so that alternative (2) would be found. The Case version also does not automatically evaluate all Program Module and File Designs. A partial set of Program Modules and File designs (~50,000) is generated. A smaller number of designs (~200 for the Company Y example) must be selected manually and then input to SODA to be evaluated. Procedures have been developed²⁰ to eliminate the need for man-machine interaction in SODA/ALT and the procedures are being implemented in the Purdue version.

The program has been run using an example problem called Company "Y". The example consists of 117 Processes and 180 Data Sets. Approximately 30 runs were required to debug the problem statement for Company Y.

A single run for SODA/PSA takes about 120 seconds of execution on the UNIVAC 1108. The total time required for the Company Y example is difficult to estimate since the SODA program was not run from beginning to end at one time; many of the submodels were run, then a data file passed to the next submodel and so on.

A series of hypothetical computers is described in the hardware file. The hardware file consists of 3 CPU's with 5 core options for each CPU. The auxiliary memory option consists of two types of tape drives and two types of disk units.

ACKNOWLEDGMENTS

The author is indebted to Mr. Hasan Sayani, Mr. Carl Singer and Dr. Daniel Teichroew of the University of Michigan for their contribution to the development of SODA.

REFERENCES

- 1) BÖRJE LANGEFORS, Some Approaches to the Theory of Information Systems BIT 3, 4, 1963.
 - 2) BÖRJE LANGEFORS, Information System Design Computations using Generalized Matrix Algebra BIT 5, 2, 1965.
 - 3) J. F. NUNAMAKER, JR. and A. B. WHINSTON, Design of a Corporate Computer System, Computer Sciences Department TR52, Purdue University, Lafayette, Indiana, February 1971.
 - 4) LIONELLO LOMBARDI, Theory of Files, Proceedings of the Eastern JCC. 1960.
 - 5) DANIEL TEICHROEW, ISDOS--A Research Project to Develop Methodology for the Automatic Design and Construction of Information Processing Systems ISDOS Working Paper Number 1, Case Institute of Technology, August 1967.
 - 6) WILLIAM STEIGER, Survey of Basic Processing Function: Literature and Related Topics and (2) Derivation of Primitives for Data Processing ISDOS Working Paper Number 11, Case Institute of Technology, May 1967.
 - 7) D. J. HERMAN AND F. H. HIRER, The Use of a Computer to Evaluate Computers, AFIPS Conference Proceedings, Spring Joint Computer Conference, Volume 25, Pages 383-395, 1964.
 - 8) L. ROWELL HUESMANN and ROBERT P. GOLDBERG, Evaluating Computer Systems through Simulation, The Computer Journal, Volume 10, Number 2, August 1967.
 - 9) Digest of the second conference on applications of simulation sponsored by SHARE/ACM/IEEE/SCI, New York, December 1968.
 - 10) R. B. BRIGGS, A Mathematical Model for the Design of Information Management Systems (M.S. Thesis, Division of Natural Science), University of Pittsburgh, 1966.
 - 11) T. C. LOWE, Analysis of Boolean Program Models for Time Shared, Paged Environments, Communications of the ACM, Volume 12, Number 4, April 1969.
 - 12) T. C. LOWE, Automatic Segmentation of Cyclic Program Structures Based on Connectivity and Processor Timing, Communications of the ACM, Volume 12, Number 1, January 1970.
 - 13) C. V. RAMAMOORTHY, Analysis of Graphs by Connectivity Considerations, Journal of the Association for Computing Machinery, Volume 13, Number 2, April 1966.
 - 14) R. B. MARIMONT, A New Method of Checking the Consistency of Precedence Matrices, Journal of the ACM, Volume 6, Number 2, April 1959.
 - 15) S. WARSHALL, A Theorem on Boolean Matrices, Journal of the ACM, Volume 9, Number 1, January 1962.
-

- 16) N. F. SCHNEIDEWIND, Analytical Model for the Design and Selection of Electronic Digital Computing Systems. D. B. A. Dissertation, University of Southern California, 1965.
- 17) H. E. THIESS, Mathematical Programming Techniques for Optimal Computer Use. Proceedings ACM 20th National Conference, 1965.
- 18) WILLIAM A. MCCUSKEY, On Automatic Design of Data Organization. AFIPS Conference Proceedings Fall Joint Computer Conference, Volume 37, 1970.
- 19) J. F. NUNAMAKER, JR., On the Design and Optimization of Information Processing Systems. Ph.D. Dissertation, Case Western Reserve University, 1969.
- 20) J. F. NUNAMAKER, JR., SODA: Systems Optimization and Design Algorithm. Computer Sciences Department TR51, Purdue University, Lafayette, Indiana, February 1971.